2008

Fibonacci Modeling of Malware Propagation

Yu Zhang

Bharat Bhargava
Purdue University, bb@cs.purdue.edu

Report Number:
08-017

Fibonacci Modeling of Malware Propagation

Yu Zhang
Bharat Bhargava

CSD TR #08-017
June 2008

Fibonacci Modeling of Malware Propagation

Yu Zhang, *zhangyu@cs.purdue.edu* and Bharat Bhargava, *Fellow, IEEE, bb@cs.purdue.edu* *

***Abstract*—Self-propagating malware spreads over the network quickly and automatically. Malware propagation should be modeled accurately for fast detection and defense. Existing malware propagation models fail to consider a number of issues. First, the malware can scan a host for multiple vulnerabilities on multiple ports. Second, the vulnerability scanning can be done by multiple threads concurrently. Third, the exploitation of vulnerabilities and the infection of vulnerable hosts cannot be done instantly. Fourth, the malware propagation can start from multiple places in the network rather than a single release point. Finally, the malware copies can collaborate with each other to cause much more damages.**

Inspired by the Fibonacci Number Sequence, We develop the discrete-time Generic Fibonacci Malware Propagation (GFMP) Model. We present the malware propagation tree and forest, prove that the GFMP model is analytically sound, and compare it to existing models. To our knowledge, the GFMP model is the first model that considers and quantifies issues of multi-port scanning, multi-threading, infection time, multiple starting points, and collaboration among infected hosts. Experimental results show that the GFMP model can accurately represent the malware propagation, and fits in the propagation data of the real-world malware such as the notorious Witty Worm.

***Index Terms*—Fibonacci, Propagation, Network Security.**

I. Introduction

Malware is the software designed to compromise computer systems. Examples of malware include Logic Bomb, Virus, Worm, and Botnet [2], [7]. Malware can be classified into two categories: self-propagating malware and non-self-propagating malware. Self-propagating malware poses a much more serious threat due to its ability to propagate through networks to infect a large number of hosts. For instance, worms, which are famous examples of the self-propagating malware, had infected thousands of computers [1], [10], [11], [15]. Malware replicates itself and intrude vulnerable hosts without human intervention. Malware can carry malicious payloads that can be released upon infection of the vulnerable hosts. Malware can cause significant damages, including consumptions of network bandwidth, destructions of infected hosts, and leak of private information such as credit card numbers, etc.

The malware propagation consists of four steps:
1. *Reconnaissance*: search vulnerable victim hosts by performing port scans;
2. *Infection*: transmit malicious payloads, exploit vulnerabilities on victim hosts to gain control;
3. *Discovery*: perform information-gathering activities on victim hosts, e.g., stealing passwords and personal files;
4. *Destruction*: perform destructive activities on victim hosts, e.g., formatting the hard disk.

* Authors are with Department of Computer Sciences, Purdue University, West Lafayette, IN, 47906.

After the second step (infection), the malware is ready to propagate from the newly infected host to another by repeating the whole process.

The first step of malware propagation is Reconnaissance, during which the malware discovers the vulnerable victims. Reconnaissance is normally done by performing port scans. To perform a thorough port scan, the malware sends probe packets to each port on each victim host, and analyzes the responses from victim hosts. In a hypothetical scenario, a packet sent to FTP port 21 on the victim host triggers a reply packet, which is then analyzed by the malware to infer detailed information such as the type and version of the operating system about the victim host. Based on these information, a more-informed attack can be launched (e.g., exploiting the vulnerability that exists on the particular operating system). The malware has to perform port scans for a huge number of IP address/port number combinations. In IPv4 networks, the size of IP address space is 2^{32} and the size of port number space is 2^{16} [14]. Hence, the size of search space for IP address/port number combination is 2^{48}. While the large size of search space renders port scanning a daunting task, malware authors have employed sophisticated techniques to perform fast scanning. For example, many real-world worms search vulnerabilities only on a particular port, which effectively reduces the size of search space to 2^{32} [9], [19], or commonly used ports (e.g., FTP port 21 and HTTP port 43).

It is clear that malware with different scanning and propagation strategies have different spread time. A number of models have been proposed to characterize the propagation of worms, including the Analytical Active Worm Propagation (AAWP) model and the epidemiological two-factor model [1], [10], etc. We observe that existing malware propagation models fail to take into consideration a number of issues:

- That the malware can scan a host for multiple vulnerabilities. For instance, while the malware may fail to find any vulnerability on the FTP port 21 of a host, such failure does not exclude the possibility that the host has other vulnerabilities. Therefore, sophisticated malware look for vulnerabilities on other ports as well, e.g., the DNS port 53. In the case that the malware discovers multiple vulnerabilities, it has the option to exploit the best according to some criteria (e.g., infection time).
- That the scanning can be done by multiple threads. Multi-threaded malware can scan and infect multiple machines concurrently. Moreover, since vulnerabilities exist on many ports, multi-threaded scanning of multiple ports on one host is an effective way to expedite port scans. Most existing models, including the AAWP model, fail to consider that the malware may spawn a large number of threads to scan concurrently.
- That the exploitation of vulnerabilities and the infection

of victim hosts are not done instantly. It takes time for the malware to transmit the payload, exploit a vulnerability, and subvert the defense system on the victim host. A newly found vulnerable host can neither be infected immediately nor be ready to infect other hosts. Although the AAWP model claims to be able to incorporate the infection time, it simply makes the clock ticks larger, without precise calculation of the ratio of scan time to infection/propagation time. Note, however, that port scans can be done much faster than infections. In the extreme case, Figure 1.(c) in [1] assumes that the infection time could be as long as 60 seconds, while the scanning time for one IP/port combination is usually shorter than 0.1 second [25]. We discuss this issue further in Section V-B.

• That the malware propagation can start from multiple places in the network rather than a single starting point, and the infected hosts can collaborate to cause much more damages (e.g., the Botnet [7] and the orchestrated attacks to Estonia [32]). For example, multiple attackers can simultaneously release the same malware at multiple places, and researchers suspect the Witty Worm [2] was released from multiple IP addresses at the same time. The malware can be released in different geographical regions of the world, e.g., Europe, Asia, and North America, which will significantly expedite its propagation. Although commonplace in the real world, multiple starting points and collaboration among attackers are not represented by existing models.

To address these issues, we propose the discrete-time Generic Fibonacci Malware Propagation (GFMP) Model, which is inspired by the Fibonacci Number Sequence. In the Fibonacci rabbit problem, newly-born rabbits cannot give birth to baby rabbits immediately. Instead, they need some time to get mature, which is reminiscent of the infection/propagation time problem discussed above: similarly, a host cannot scan and infect other hosts until maturity, i.e., completely infected.

The GFMP model employs the malware propagation tree and forest to represent the infection and propagation time. We prove that the GFMP model is analytically sound, derive its important properties, and compare it to existing models. To our knowledge, the GFMP model is the first model that considers and quantifies issues of multi-port scanning, multi-threading, infection time, multiple starting points, and collaboration among infected hosts. Experimental results show that our model can accurately represent the malware propagation, and fits in the propagation data of the real-world malware such as the notorious Witty Worm [2].

The rest of the paper is organized as follows. Section 2 discusses related work. In Section 3, we briefly review the Fibonacci Number Sequence and infer several important properties. In Section 4, we present the propagation tree and propagation forest. In Section 5, we describe the GFMP model and its properties, and compare it to existing models. In Section 6, we conduct experiments to evaluate the GFMP model. Section 7 concludes our research.

II. Related Work

Scan Strategy. Over the years, researchers have proposed various scanning algorithms for the malware, including: *naive random scanning*, in which the malware chooses a random address uniformly from the IPv4 address space [1]; *localized scanning*, in which the malware scans a local IP address (e.g., addresses that are in the same subnet as the malware) with a high probability p and scans a random address with a low probability (1-p) each time [30]; *importance scanning*, in which the malware assumes that the vulnerable hosts are unevenly distributed and such distributions are obtainable [6]; *self-learning scanning*, in which the malware estimates the distribution of the vulnerable hosts [27]; *hit-list scanning*, in which the malware uses an existing list. e.g., BGP routing table list, social network list, etc., to look for vulnerable hosts [15]; *permutation scanning*, in which the malware could determine whether a host was infected or not and change scan targets [15]; *sampling scanning*, in which the malware first samples the target network and then spread to the corresponding subnet [3]; and *passive scanning*, in which the malware analyzes the network traffic passively without sending probe packets, etc.

Collaboration. Malware can collaborate with each other and perform much more efficient reconnaissance. S. Staniford et al. [15] discussed the Warhol worm, which propagates extremely quickly by self-coordination with both hit-list scanning and permutation scanning. B. Wiley [16] described an abstract distributed and collaborative worm Curris Yellow. C. Gates [29] discussed possible collaborations in port scans, and how to detect collaborative port scans.

Port Scan. Port scanners can employ various techniques to perform scanning. Examples include: SYN Scan, in which the scanner produce its own IP packet and sends TCP SYN packets to the victim hosts and analyzes the responses; UDP scan, in which the scanner sends UDP packets to the victim hosts and checks whether ICMP port unreachable messages are received afterwards; and application-layer scans, such as HTTP/FTP/DNS scans [19], [12]. Port scanning can be performed on multiple ports. Famous ports include port 21 for FTP, port 53 for DNS, port 80 for HTTP, port 25 for SMTP, and port 443 for HTTPs [14]. Some scanners perform the scanning in two-iterations: they scan with one technique, e.g., the SYN scan, first, and scan the un-denied ports with another technique. For instance, the famous NMAP scanner [25] uses the two-iteration approach when executed with the -SUV option.

IPv6 Scan. J. Yang [13] discussed how to defend worms in IPv6 networks. S. Bellovin et al. [23] presented worm propagation strategies for IPv6 networks. A. Kamra et al. [12] proposed a DNS-scan method that can achieve high spread rates in IPv6 networks. In general, IPv6 networks have much larger address space, rendering the port scan more difficult. However, some IPv6 features reduce the address space, and the malware can utilize the high-speed network connection in the future to speed up the propagation.

Malware Propagation. A. Wagner et al. [24] presented characteristics of worms, including the protocol, the size of the payload, and the scanning strategy, etc. C. Zou [9] et al. analyzed the performances of different worm propagation strategies. A.G. Voyiatzis et al. [20] described a class of worms that target network components such as routers. M. Vojnovic et al. [3] discussed how to minimize the required number of

TABLE I
NOTATIONS USED IN THIS PAPER

Notation	Explanation
b	the number of IP addresses on the blacklist of the malware
c	the number of ports the malware scans for each IP address
conh	the number of contagious hosts that can infect other hosts
d	destruction rate: the number of destructed hosts over the number of infected hosts
k	the number of threads in the malware
p	patching rate: the rate at which the vulnerable machines are patched
r	birth rate: the rate at which the new vulnerable hosts joins the network
v	the number of vulnerable (excluding infected) host/port combinations
V	the number of vulnerable (including infected) host/port combinations
x/y	the initial values for the FNS/LNS
ϕ	see Equation (2) for the formal definition. (For FNS, ϕ is approximately 1.618, the golden ratio.)
θ	see Equation (8) for the formal definition. (For FNS, θ is $\sqrt{5}$.)
α/β	the coefficients for the LNS. See Equation (7).
PT	Propagation Time
IT	Infection Time

scans to infect hosts. Storm Worm [4], [5] used the Distributed Hash Table (DHT) protocol based on Kademlia [26] to control the infected nodes. Z. Chen et al. [1] proposed the Analytical Active Worm Propagation (AAWP) model. C. Zou et al. [10] proposed the epidemiological two-factor model.

Defense. J. Wu et al. [17] proposed a worm detection architecture for various worm scanning techniques. J. Twycross et al. [18] built a virus throttle program that can detect the malware based on their abnormal network behaviors. J. Jung et al. [21] developed the Threshold Random Walk (TRW) algorithm to identify malicious remote hosts. A. Kumar et al. [28] presented the analysis of the Witty Worm and inferred about the IP address where the Witty Worm was released. S. Staniford et al. [22] described Spice, a port scanner that can detect stealthy scans.

III. Background on Fibonacci Number Sequence

In this section, we briefly summarize the Fibonacci Number Sequence (FNS) and discuss its generalizations. We infer several important properties of the FNS and discuss their uses in the malware propagation. In Section V we discuss in detail on how the FNS is applied to analyze the malware propagation and to model multi-threading, infection time, multiple start points, and collaborative attacks. The FNS is named after Leonardo Fibonacci. Interested readers are referred to [31] for thorough discussions of the FNS. Table I lists the notations used throughout this paper.

A. Fibonacci rabbit problem

We briefly present the famous Fibonacci rabbit problem: *In the beginning there is no rabbit. After one month, one pair of baby rabbits is brought in. The baby rabbits will get mature after one month. Each pair of mature rabbits gives birth to a new pair of baby rabbits every month. The problem is: how many pairs of rabbits will there be after n months?*

B. Definition of Fibonacci Number Sequence

To solve the fibonacci rabbit problem, we assume that rabbits never die. We use F_n to represent the number of pairs of rabbits there are after n months. Note that F_0 = 0 and F_1 = 1. We observe that $F_2 = 1 \neq 2$, since after 2 months the first pair of baby rabbits will get mature and cannot yet give birth to new baby rabbits. Note that in the malware propagation field, most existing models ignored this issue.

Since rabbits never die, to calculate how many pairs of rabbits there are after n (n>1) months, we simply add the newly born rabbits to the existing rabbits after (n-1) months, which is represented by F_{n-1}. Not all those F_{n-1} pairs of rabbits are mature. Because the baby rabbits take one months to get mature, we observe that the baby rabbits are those born within one month, i.e., the (n-2, n-1) month window. Therefore, the rabbits that were born before this window are all mature by Month n. There are F_{n-2} pairs of such rabbits.

Assume that F_{n-1} and F_{n-2} are known. We have:
$F_n = F_{n-1} + F_{n-2}$ when $n > 1$.

Hence, the solution can be summarized as:

$$F_n = \begin{cases} 0 & \text{if n = 0;} \\ 1 & \text{if n = 1;} \\ F_{n-1} + F_{n-2} & \text{if n > 1.} \end{cases} \quad (1)$$

We call the numbers generated by the recursive definition (1) Fibonacci numbers, and call the number sequence FNS.

C. Properties of Fibonacci Number Sequence

1) Closed-Form Expression: We can solve the recursive equation of the FNS with the initial conditions F_0 = 0 and F_1 = 1.

$$F_n = \frac{\phi^n - (1-\phi)^n}{\theta}, \quad where \quad \theta = \sqrt{5} \quad and \quad \phi = \frac{1+\theta}{2} \quad (2)$$

Since $|\frac{(1-\phi)^n}{\theta}|$ is a very small number (smaller than 0.1 when n is larger than 3), we can safely discard it and rewrite the result as:

$$F_n = \frac{\phi^n}{\theta}, \quad where \quad \theta = \sqrt{5} \quad and \quad \phi = \frac{1+\theta}{2} \quad (3)$$

Note that ϕ is the golden ratio (approximately 1.618).

2) Growth Rate: The growth rate of the FNS, regardless of the initial values (except for F_0 = F_1 = 0), is:

$$\lim_{n\to\infty} \frac{F_{n+1}}{F_n} = \phi$$

Hence, the FNS approximately follows the exponential growth at the rate of the golden ratio ϕ when n is large. Note that the malware propagation is also exponential before saturation [4],[30].

D. Generic Fibonacci Number Sequence: Arbitrary Initialization

1) Definition of Generic Fibonacci Number Sequence: If the initial values of the FNS are changed to x and y respectively, we will have the generic FNS $G_{x,y,n}$:

$$G_n = \begin{cases} x & \text{if n = 0;} \\ y & \text{if n = 1;} \\ G_{n-1} + G_{n-2} & \text{if n > 1.} \end{cases} \quad (4)$$

2) How to Calculate Generic Fibonacci Number Sequence: $G_{x,y,n}$ can be represented by the original FNS.

$$G_{x,y,n} = xF_{n-1} + yF_n \quad (5)$$

Due to space limitations, we omit the proof for Equation (5). Note that F_{-1} = F_1 - F_0, thus Equation (5) still holds when n is 0.

Similar to Section III-C1, when n is larger than 3, we apply Equation (3) and rewrite $G_{x,y,n}$ as:

$$\begin{aligned} G_{x,y,n} &= xF_{n-1} + yF_n \\ &= x\frac{\phi^{n-1}}{\theta} + y\frac{\phi^n}{\theta} \\ &= (\frac{x}{\phi} + y)\frac{\phi^n}{\theta} \\ &= (\frac{x}{\phi} + y)F_n \end{aligned} \quad (6)$$

We observe that the Generic FNS can be approximately calculated by multiplying the original FNS by a constant factor.

3) The Shift Property of Generic Fibonacci Number Sequence: Given Equation (6) we can infer the "Shift" property of the Generic FNS, i.e., the generic Fibonacci number $G_{x,y,n}$ can be represented by the original Fibonacci number of F_{n+s}, where s is the number of shifts and s is equal to $\frac{\log(y\phi+x)}{\log\phi} - 1$. Formally:

Theorem 1: $G_{x,y,n}$ = $F_{[n+\frac{\log(y\phi+x)}{\log\phi}-1]}$
Proof: According to Equation (6),

$$\begin{aligned} G_{x,y,n} &= (\frac{x}{\phi} + y)\frac{\phi^n}{\theta} \\ &= (\frac{x+y\phi}{\phi})\frac{\phi^n}{\theta} \\ &= (\frac{\phi^{\log_\phi(x+y\phi)}}{\phi})\frac{\phi^n}{\theta} \\ &= (\phi^{[\frac{\log(x+y\phi)}{\log\phi}-1]})\frac{\phi^n}{\theta} \\ &= \frac{\phi^{[n+\frac{\log(x+y\phi)}{\log\phi}-1]}}{\theta} \end{aligned}$$

$$Apply \quad Equation(3), = F_{[n+\frac{\log(y\phi+x)}{\log\phi}-1]} \quad \blacksquare$$

Given x and y, the number of shifts s is a constant number. Theorem 1 has important implications on the Fibonacci malware propagation: it quantifies the effects of different initialization values, and proves that the same effects can be achieved by "shifting" the index of the original FNS by a constant number. Hence, the effects of hitlist scanning and flash scanning, etc., can be quantified in the model by shifting the regular scanning. We discuss this further in Section V.

4) The Linear Property of Generic Fibonacci Number Sequence: The Linear Property of the Generic FNS states that the sum of two Generic FNSes with initial values (x_1,y_1) and (x_2,y_2) is equivalent to the Generic FNS with the initial values (x_1 + x_2, y_1 + y_2, respectively. Formally:

Theorem 2: $G_{x_1,y_1,n}$ + $G_{x_2,y_2,n}$= $G_{x_1+x_2,y_1+y_2,n}$
Proof: According to Equation (6),

$$\begin{aligned} G_{x_1+x_2,y_1+y_2,n} &= (\frac{x_1+x_2}{\phi} + y_1 + y_2)\frac{\phi^n}{\theta} \\ &= (\frac{x_1}{\phi} + y_1)\frac{\phi^n}{\theta} + (\frac{x_2}{\phi} + y_2)\frac{\phi^n}{\theta} \\ &= G_{x_1,y_1,n} + G_{x_2,y_2,n} \end{aligned} \quad \blacksquare$$

Corollary 1: $G_{mx,my,n}$ = m$G_{x,y,n}$
Proof: According to Theorem 2,

$$\begin{aligned} G_{mx,my,n} &= G_{x,y,n} + G_{(m-1)x,(m-1)y,n} \\ &= 2G_{x,y,n} + G_{(m-2)x,(m-2)y,n} \\ &= ... \\ &= kG_{x,y,n} + G_{(m-k)x,(m-2)y,n} \\ &= ... \\ &= mG_{x,y,n} \end{aligned} \quad \blacksquare$$

E. Generic Lucas Number Sequence

A further generalization of the FNS is the Generic Lucas Number Sequence (LNS). Given constant integers x and y, we have:

$$H_n = \begin{cases} x & \text{if n = 0;} \\ y & \text{if n = 1;} \\ \alpha H_{n-1} - \beta H_{n-2} & \text{if n > 1.} \end{cases} \quad (7)$$

The Generic FNS is a special case of the Generic LNS when α = 1 and β = -1. To investigate malware propagation, we are interested in the case where α = 1 and β = -q ($|q|<\frac{1}{4}$). We discuss this further in Section V-C.

When α = 1 and β = -q ($|q|<\frac{1}{4}$), as in Section III-C1, we can solve the recursive equation of the special LNS with the initial conditions H_0 = x = 0 and H_1 = y = 1, and get its closed-form expression:

$$\begin{aligned} H_n = \frac{\phi^n - (1-\phi)^n}{\theta}, where \quad \theta = \sqrt{1+4q} \\ and \quad \phi = \frac{1+\theta}{2} \end{aligned} \quad (8)$$

Since $|q|<\frac{1}{4}$ and 4q<1, we can expand θ using binomial expansion:

$$\begin{aligned} \theta &= \sqrt{1+4q} \\ &= \sum_{m=0}^{+\infty} \frac{(-1)^n(2n)!}{(1-2n)(n!)^2 4^n}(4q)^m \\ &\approx \sum_{m=0}^{n} \frac{(-1)^n(2n)!}{(1-2n)(n!)^2 4^n}(4q)^m, n = 2 \\ &= 1 + \frac{4q}{2} - \frac{(4q)^2}{8} \\ &= 1 + 2q - 2q^2 \end{aligned}$$

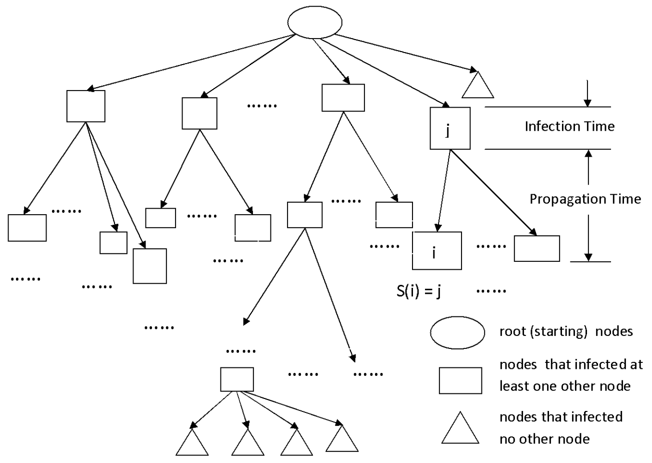
j
Infection Time
Propagation Time
i
S(i) = j
root (starting) nodes
nodes that infected at least one other node
nodes that infected no other node

Fig. 1. The propagation tree of the self-propagating malware.

Hence, given that $|q|<\frac{1}{4}$, $|\frac{(1-\phi)^n}{\theta}|$ = $|\frac{(\phi-1)^n}{\theta}|$ = $|\frac{(\frac{\theta-1}{2})^n}{\theta}|$ = $|\frac{(q-q^2)^n}{1+2q-2q^2}|$ is a very small number. Thus, we can can safely discard it and rewrite the result as (when α = 0 and β = -1):

$$H_n = \frac{\phi^n}{\theta}, \quad where \quad \theta = \sqrt{1+4q} \quad and \quad \phi = \frac{1+\theta}{2} \tag{9}$$

Note that when q = 1, we get the closed-form expression for FNS as in Section III-C1. We observe that Equation 3 and Equation 9 differ only in the constants. Therefore, the properties, including the Shift Property and the Linear Property, of the Generic FNS all hold for the Generic LNS when α = 1 and β = -q (q<$\frac{1}{4}$). Due to space limitations, we omit the formal proof for this observation.

IV. Propagation tree and forest

A. The propagation tree of self-propagating malware

As shown in Fig. 1, assume that the malware propagation starts from a single node, We develop the propagation tree of the self-propagating malware $PropT_r$, which consists of:
1. the root node r: the node where the the malware author releases it;
2. the intermediate nodes: the nodes that caused direct infections of at least one other node; and
3. the final nodes: the nodes that caused no direct infections of other nodes.
Note that the final nodes may have attempted to infect other nodes, although those attempts, if any, must be unsuccessful.

Formally, we define:
1. the source (parent) function S, such that:
S(i) = j, iff node i, j $\in PropT_r$, and j is a parent of i in the tree $PropT_r$. As shown in Fig. 1, if S(i) = j, node i is the child of node j.

2.The propagation tree of the self-propagating malware $PropT_r$ is a directed tree in which each node is either:
1. the root node r, where $\nexists$ node j $\in PropT_r$ such that j $\neq$ r and S(r) = j;
2. the intermediate node i, where $\exists$ node j $\in PropT_r$ such that j $\neq$ i and S(j) = i; or
3. the final node f, where $\nexists$ node j $\in PropT_r$ such that j $\neq f$ and S(j) = f, and $\exists$ node k $\in PropT_r$ such that k $\neq$ f and S(f) = k.

B. The propagation forest of self-propagating malware

If the malware is released at k sources, r_i, i $\in$ [1..k], we can generate one propagation tree for the malware propagation rooted at each source node. These propagation tress form the propagation forest.

The propagation forest of self-propagating malware F_{prop} is the disjoint union of the propagation trees rooted at nodes r_i, i $\in$ [1..k], formally:

$$F_{prop} = \bigcup_{i=1}^{k} PropT_{r_i}$$

C. The infection time and propagation time

As shown in Fig. 1, for the newly infected victim host, there is a short delay between the intrusion of the malware and the propagation of the malware to other hosts. Such delay includes the time spent on the exploitation of the vulnerability on the victim host and subversion of the victim host. We denote the delay as infection time.

Intuitively, we define the Infection Time (IT) as the time interval between the start of infection on a particular host (i.e., the time when the host was initially intruded) and the start of propagation on the same host (i.e., the time when the same host was starting to infect other hosts). Formally,

$$IT = T_{StartPropagation} - T_{StartInfection} \tag{10}$$

Note that actual infection time may vary and follow particular probability distributions.

At the same time, we could define Propagation Time (PT) of the malware between two hosts as: the time interval between the infection of a particular host (denote it as s) and the successful infection of a subsequent target host (denote it as T(s)) that was caused by this particular host. Formally,

$$PT = T_{Infection(T(s))} - T_{Infection(s)} \tag{11}$$

For a host m that neither received intrusion attempts nor was infected successfully, the time of infection ($T_{infection(m))}$) is defined as +∞ (infinite).

We can measure the propagation time for all infected hosts and collect the statistics about them, e.g., we can calculate the average propagation time. However, there is one problem with definition (2): it works only if there is at least one subsequent successful infection from the original host (S). If the infection attempt was unsuccessful (e.g., if the target host was invulnerable) or there was no subsequent infection attempt (e.g., if the malware on the host was quarantined by administrators) the propagation time is +∞ (infinite).

Alternatively, we can calculate the propagation time from the infected hosts, under the observation that each infected host must be infected by some source host. Hence, we define the Propagation Time (PT) of the malware between a host and its infector as: the time interval between the successful infection of a particular host (denote it as t) and the infection of the host that caused the infection of this particular host [8] (denote it as S(t)). Formally,

$$PT(S(t), t) = T_{Infection(t)} - T_{Infection(S(t))} \tag{12}$$

We can infer several important properties for Propagation Time (PT).

1. Additivity: if there are three hosts (Host m, n, and o) that satisfy the following two conditions:

a) the malware propagated directly from Host m to n; and
b) the malware propagated directly from Host n to o,
then the propagation time between Host m and o is the sum of the propagation time between Host m and n and the propagation time between Host n and o. Formally,

For Hosts m, n, and o that satisfy S(o) = n and S(n) = m:

$$PT(m, o) = PT(m, n) + PT(n, o), \tag{13}$$

2. Diameter: the diameter of the tree (Diameter($PropT_r$))is the time elapsed since the release of the malware until the infection of the last vulnerable hosts (denote it as lv). Hence, we can use PT(r, lv) to represent the diameter of the tree: Diameter($PropT_r$) = PT(r, lv).
Assume that there are n intermediate nodes on the path between r and lv. We denote them as $node_i$, i $\in$ [1..n], where:

$$\begin{cases} S(node_i) = r, & \text{if i = 1;} \\ S(lv) = node_i, & \text{if i = n;} \\ S(node_{i+1}) = node_i & \text{if i} \in \text{(1..n).} \end{cases} \tag{14}$$

Using Property 1, Diameter(T_{prop}) can be further calculated as:

Diameter($PropT_r$)
= PT(r, lv)
= PT(r, $node_i$) + PT($node_1$, $node_2$) + ... + PT($node_i$, $node_{i+1}$)
+ ... + PT($node_n$, lv)

$$= PT(r, node_i) + PT(node_n, lv) + \sum_{\substack{i \\ \mathbf{i \in (1..n)}}} PT(node_i, node_{i+1}) \tag{15}$$

V. Generic Fibonacci Malware Propagation (GFMP) Model

In this section, we describe the discrete-time Generic Fibonacci Malware Propagation (GFMP) model, and show how we apply Fibonacci numbers to model the malware propagation and address the issues presented in the previous sections.

A. Preliminaries

We assume that during the Reconnaissance step the malware will perform port scanning to discover the vulnerable ports on the target host.

Malware may selectively scan the IP addresses rather than scan all. E.g., reverse engineering [30], [11] shows that Code Red I and II never scan some IP addresses, including the local (127.0.0.0/8) and multicast (224.0.0.0/8) addresses. This fact is often overlooked by researchers (e.g., in [10] the authors assume that CodeRed will scan all IP addresses with equal probability). Assume that the malware puts b IP addresses on its blacklist, i.e., it will never scan those IP addresses, and IPv4 is in use, the number of the IP addresses the malware may scan is (2^{32} - b).

We further assume that the malware scans c ports for each destination IP address. Hence, the total search space for the malware is c(2^{32} - b).

Advanced real-world scanners are mostly multi-threaded. Most existing models overlooked multi-threading issues in modeling the malware propagation. In our model, we assume that the malware employs multi-threaded programming and scans multiple address/port combinations concurrently. If there are k threads of each malware scanning module,we assume that each time each infected host can scan k address/port combinations. We do not assume that these k scans are independent.

Hence, we need to calculate how many new vulnerable IP address/port combinations are discovered each time. Note that vulnerability discovery is not the same as successful infection. After the vulnerability discovery, the malware still needs some time to propagate to the victim host and exploit the vulnerability. We assume that there are v uninfected vulnerable IP address/port combinations (multiple ports on one host can be infected) and i infected hosts in the network before the scanning. Given that each time each infected host can perform k scans simultaneously, we would like to know how many out of those v combinations will be discovered by all infected hosts.

We denote the number of contagious hosts as conh (contagious host). The number of new vulnerable IP address/port combinations discovered (denoted as *newlyinfected*) during the scanning is:

newlyinfected
= v $\times$ (the probability that a given IP address/port combination will be discovered by at least one of the conh infected hosts)
= v $\times$ (1 - the probability that a given IP address/port combination will not be discovered by all the conh infected hosts)
= v $\times$ (1 - (the probability that a given IP address/port combination will not be discovered by one infected host)^{conh}]
= v $\times$ [1 - (1 - The probability that a given IP address/port combination will be discovered by one infected host))^{conh}]

Since the probability that a given IP address/port combination will be discovered by one infected host is $\frac{k}{c(2^{32}-b)}$, we have:

$$\textit{newlyinfected} = v \times [1 - (1 - \frac{k}{c(2^{32} - b)})^{conh}] \tag{16}$$

B. More on Infection Time and Propagation Time

As discussed in Section IV-C, we cannot ignore the infection time and propagation time in the malware propagation model.

The epidemiological models use the infection time and the propagation time as parameters in simulations without detailed discussions [10]. In the AAWP model [1], all infected hosts (including the ones that were newly infected) perform their scanning activities at the next time tick (denote it at t and the length of the time tick as L(t)). Therefore, the the newly infected hosts that were infected between (t, t + L(t)) are treated equally and the hosts infected near time t will perform the same number of scans as the hosts infected near time t + L(t). Such equal treatment is imprecise. The AAWP model also models the infection time by simply making the clock ticks larger without considering the difference between the scan time and infection/propagation time. The scan can usually be performed much faster with advanced scanning techniques than the infection. In the extreme case, the scan time for one IP address/port combination usually takes less than 0.1 seconds [25], while [1] assumes that the infection could be as large as 60 seconds.

Moreover, the Theorem 1 for the AAWP model is proved by induction on the number of scans, which can either be successful (which brings a newly infected host) or unsuccessful (which brings no new host). Hence, each induction step will either add a host or do nothing. At the next time tick, the number of infected host will be either unchanged or increased by one. Hence, The AAWP model assumes that the scans are performed step by step by the worms, i.e., in each step the scan of one worm is performed, and the number of the infected hosts is updated. This is not always true, e.g., according to its user documentation, the famous NMAP scanner [25]: is capable of scanning many hosts in parallel by dividing targets into multiple groups, and scanning one group (e.g., 50 hosts) at a time.

To our knowledge, the AAWP model does not take into consideration the propagation time of the worm, and ignores the time of data transfer between hosts.

C. Generic Fibonacci Malware Propagation (GFMP) Model

In the Fibonacci rabbit problem, the newly born rabbits are not mature, and cannot produce rabbit right away. Instead, they take some time to get mature, which is reminiscent of the infection/propagation time problem in the malware propagation modeling.

Inspired from the Fibonacci Number Sequence (FNS), we propose the Generic Fibonacci Malware Propagation (GFMP) Model.

In the GFMP model, A malware cannot scan or infect other hosts until it has gained control of the infected host. We use the propagation time to model the time delay between the time when the host gets attacked and the time when the host starts to attack other hosts.

1. We first derive the formula of the GFMP Model. We denote the total number of vulnerable hosts in the beginning as V and the number of infected hosts as I_j, where j denotes the time tick (one time tick could represent one second). We denote the length of the time tick as L(t). We assume that the administrators may patch the vulnerable hosts. We assume that the propagation time (PT) defined in Section IV is two time ticks for all infections. Hence, the newly infected hosts intruded at time t are not able to infect new hosts at time t + L(t), but will be able to infect new hosts at time t + 2×L(t). At time tick j + 2, there are I_j infected hosts that are contagious and can infect other hosts. Formally:

$$conh = I_j \tag{17}$$

At time tick j+1, the number of uninfected vulnerable hosts is the number of all unpatched vulnerable (including infected and newly born) hosts minus the number of infected vulnerable hosts. Note that neither dead infected hosts nor newly born hosts could be patched. We assume that the malware can carry destructive payloads (e.g., formatting the hard disk). In this case, the destructed hosts are wiped out, and must be removed from the vulnerable host list. We define destruction rate to be the number of destructed hosts divided by the number of infected hosts, considering that only infected hosts can be destroyed. Therefore, we calculate the dead hosts by multiplying the destruction rate to the number of infected hosts, instead of the number of all vulnerable hosts. We denote the destruction rate of the hosts as d, the birth rate of the vulnerable hosts (e.g., new vulnerable hosts that just joined the network) as r, and the patching rate of the vulnerable hosts as p. Formally:
The number of hosts that are vulnerable (including infected and newly born) and can be patched at time tick j+1 is:

$$v_{j+1} = (1-p)v_j - d * I_j + r * v_j = (1-p+r)v_j - d * I_j$$

This is a recursive calculation. We expand the recursion and get:

$$v_{j+1} = (1-p+r)^{j+1}v_0 - \sum_{k=0}^{j}(1-p+r)^k * d * I_{j-k}.$$

Given that v_0 = V, we have:

$$v_{j+1} = (1-p+r)^{j+1}V - \sum_{k=0}^{j}(1-p+r)^k * d * I_{j-k} \tag{18}$$

The number of hosts that are vulnerable but not infected is:

$$v = v_{j+1} - I_{j+1} \tag{19}$$

After one time tick (time tick j+2), without considering destruction and patching, the number of infected hosts is the sum of the number of infected hosts at the previous time tick (j+1) and the number of newly infected hosts during the time tick.
The number of infected hosts that died or were patched is

$$dp = (d+p) * I_{j+1} \tag{20}$$

The number of newly infected hosts is calculated in Section V-A.

Hence, given (16), (17), (18), (19), (20), we have:

$$\begin{aligned}
&I_{j+2}\\
&= I_{j+1} + newlyinfectedhosts - deadorpatchedhosts\\
&= I_{j+1} + v * (1 - (1 - \frac{k}{c(2^{32}-b)})^{I_j}) - (d+p) * I_{j+1}\\
&= (1-d-p)I_{j+1} + [(1-p+r)^{j+1}V -\\
&\sum_{k=0}^{j}((1-p+r)^k * d * I_{j-k}) - I_{j+1}][1 - (1 - \frac{k}{c(2^{32}-b)})^{I_j}]
\end{aligned} \tag{21}$$

Note that this recursive Fibonacci growth function applies when there is at least one vulnerable host in the system.

2. If the birth rate and the patching rate are equal, (21) can be simplified to:

$$\begin{aligned} & I_{j+2} \\ & = (1-d-p)I_{j+1} + \\ & (V - d\sum_{k=0}^{j} I_{j-k} - I_{j+1})[1-(1-\frac{k}{c(2^{32}-b)})^{I_j}] \end{aligned} \tag{22}$$

3. If the birth rate, the destruction rate, and the patching rate are all zero, (21) can be simplified to:

$$I_{j+2} = I_{j+1} + (V - I_{j+1})[1-(1-\frac{k}{c(2^{32}-b)})^{I_j}] \tag{23}$$

4. We can use binomial expansion to expand and simplify 1 - (1 - $\frac{k}{c(2^{32}-b)})^{I_j}$:

$$\begin{aligned} & 1-(1-\frac{k}{c(2^{32}-b)})^{I_j} \\ & = 1 - \sum_{m=0}^{+\infty} \binom{I_j}{m} (-\frac{k}{c(2^{32}-b)})^m \\ & = 1 - \sum_{m=0}^{n} \binom{I_j}{m} (-\frac{k}{c(2^{32}-b)})^m, n = +\infty \end{aligned} \tag{24}$$

We observe that: k represents the multi-threading level of the malware propagation scanner, and normally range from 1 to 2^{10} or one thousand; V represents the number of vulnerable hosts(including infected hosts), and is normally smaller than 2^{20} or one million; c represents the number of ports that the malware is scanning, and $c > 0$; and b represents the number of IP addresses that the malware puts on the blacklist. If the malware puts local and multicast addresses on the blacklist only, 2^{32} - b $\approx 2^{32}$. Hence, $\frac{kV}{c(2^{32}-b)} < \frac{2^{10}\times 2^{20}}{2^{32}} = \frac{1}{4}$. Note that these are conservative estimations since normally k is much smaller than 2^{10} and V is smaller than 2^{20}. Since the number of infected hosts cannot be larger than the number of all vulnerable hosts, i.e., $I_j \leq$ V, we conclude that $I_j * \frac{k}{c(2^{32}-b)}$ is small. Therefore, we can approximate (24) by setting n to 1.

When n is 1, we can rewrite (23) as:
$I_{j+2} = I_{j+1} + (V - I_{j+1})[1-(1-\frac{k}{c(2^{32}-b)})^{I_j}]$

$$\begin{aligned} & \approx I_{j+1} + (V - I_{j+1})[1 - \sum_{m=0}^{1} \binom{I_j}{m} (-\frac{k}{c(2^{32}-b)})^m] \\ & = I_{j+1} + (V - I_{j+1}) \binom{I_j}{1} (\frac{k}{c(2^{32}-b)}) \\ & = I_{j+1} + \frac{kI_j(V - I_{j+1})}{c(2^{32}-b)} \\ & = I_{j+1} + \frac{k}{c(2^{32}-b)} I_j V - \frac{k}{c(2^{32}-b)} I_j I_{j+1} \end{aligned}$$

During the initial phase of the spread of the malware, $\frac{I_{j+1}}{V}$ is a small number, we can safely throw away $-\frac{k}{c(2^{32}-b)} I_j I_{j+1}$. Therefore:

$$I_{j+2} = I_{j+1} + \frac{kV}{c(2^{32}-b)} I_j \tag{25}$$

Equation 25 suggests that the initial spread of the malware approximately follows the Generic Lucas Number Sequence (LNS) with α = 1 and β = -$\frac{kV}{c(2^{32}-b)}$:

$$I_j = \begin{cases} x & \text{if j = 0;} \\ y & \text{if j = 1;} \\ I_{j-1} - (-\frac{kV}{c(2^{32}-b)}) I_{j-2} & \text{if j > 1.} \end{cases} \tag{26}$$

As discussed above, $|\beta|$ = $\frac{kV}{c(2^{32}-b)} < \frac{1}{4}$. Therefore, the conditions specified in Section III-E that α = 1 and $|\beta| < \frac{1}{4}$ are satisfied, and we can safely apply the properties, theorems, and corollaries discussed in Section III.

5. Equation 25 holds when the propagation time PT = 2L(t) (twice as much as the length of the time tick). We now discuss the effects of different lengths of the propagation time. Generally, if PT = eL(t), where e is an integer, we have:

$$I_{j+2} = I_{j+1} + \frac{kV}{c(2^{32}-b)} I_{j+2-e}, \quad where(j+2 > e) \tag{27}$$

We observe that the longer PT is, the slower the propagation will be, which follows the intuition that longer PT delays the malicious activities of the newly infected hosts.

D. Properties of GFMP Model

We infer several properties of the GFMP Model.

1) ***Multi-threading and the Closed-Form Expression****:* Given Equation (9) and that q = $\frac{kV}{c(2^{32}-b)}$, the closed-form expression for the number of infected hosts, when x = 0 and y = 1, is:

$$\begin{aligned} I_j &= \frac{\phi^j}{\theta}, \quad where \quad \theta = \sqrt{1+4q} \quad and \quad \phi = \frac{1+\theta}{2} \\ &= \frac{[\sqrt{c(2^{32}-b)} + \sqrt{c(2^{32}-b) + 4kV}]^j}{2\sqrt{c(2^{32}-b)+4kV}[2\sqrt{c(2^{32}-b)}]^{j-1}} \end{aligned}$$

Note that the malware propagation stops when all vulnerable hosts that can be infected are indeed infected, i.e. $I_j \leq$ V. Hence, we can rewrite I_j as:

$$I_j = \begin{cases} \lambda, & \text{if } \lambda \leq \text{V;} \\ V, & \text{if } \lambda > \text{V.} \\ & (\lambda = \frac{[\sqrt{c(2^{32}-b)} + \sqrt{c(2^{32}-b)+4kV}]^j}{2\sqrt{c(2^{32}-b)+4kV}[2\sqrt{c(2^{32}-b)}]^{j-1}}) \end{cases} \tag{28}$$

In Equation (28), k denotes the number of active threads in the malware scanner. We observe that as k increases, the growth rate increases. However, we note that multi-threaded programs can easily generate huge network traffic by sending a large number of packets. Although context switching costs among the threads are smaller than those of processes, the costs increase as k increases. Real-world multi-threaded malware normally employs between 10 - 100 threads. Equation (28) was derived from Equation (25), where we assume that the number of previous infected hosts is much smaller than the number of all vulnerable hosts ($\frac{I_{j-1}}{V}$ is small), and dropped $-\frac{k}{c(2^{32}-b)}I_jI_{j+1}$. Hence, Equation (28) grows faster than the actual malware propagation when the number of infected hosts is large.

We conduct experiments to verify Equation (28). We discuss the experimental results in Section VI.

*2) **Sophisticated Scanning and the Shift Property**:* In Section V-D1, we derived the closed-form expression when the malware employ multi-threaded random scanning, and the initial values x and y are 0 and 1, respectively. However, the malware can employ more sophisticated scanning techniques, such as a combination of the scanning strategies. The malware can perform hitlist scanning to infect a large number of pre-selected vulnerable hosts first [15], then employ the regular random scanning on the newly infected hosts. The GFMP model can model such scanning strategies by different initializations of x and y. For example, if the size of the hitlist is h, we assume that at time 1 the number of infected hosts is h (the original release point of the malware) instead of 1, i.e., x = 0 and y = h.

According to the Shift Property (Theorem 1) in Section III-D3, The Generic LNS sequence determined by Equation 28 with initial values x and y can be calculated as:

$$GI_{x,y,j} = I_{[j+\frac{\log(y\phi+x)}{\log\phi}-1]} \tag{29}$$

When x = 0 and y = h, we have:

$$GI_{0,h,j} = I_{[j+\frac{\log(h\phi)}{\log\phi}-1]} \tag{30}$$

Hence, the number of infected hosts of the malware with a hitlist of size h and the combined scanning strategy at time j can be represented by the number of infected hosts of the original random-scanning malware at time (t+s), where s is the shifting number and s = $\frac{\log(h\phi)}{\log\phi}-1$.

Furthermore, according to Equation (5),

$$GI_{x,y,j} = xI_{j-1} + yI_j$$

Hence, the propagation of malware employing the combined hitlist and random scanning is the linear combination of the two propagations of malware employing merely random scanning. When x = 0 and y = h, we have:

$$GI_{0,h,j} = hI_j \tag{31}$$

We call (h) the linear Fibonacci Coefficient (FC) of the linear combination.

The real-world Witty Worm propagation confirms our analysis [2]. We discuss this issue further in the experiments section (Section VI).

*3) **Multiple Starting Points, Collaborative Attacks and the Linear Property**:* The malware propagation can start from multiple places in the network rather than a single release point, and the infected computer systems can collaborate with each other to cause much more damages. For example, the coordinated Botnet zombie nodes can collaborate to launch DoS Attacks [7] and the well-orchestrated collaborative attacks to Estonia caused large-scale disruptions [32].

We consider the representation of the following attacks:

Case 1. There are m uncoordinated attackers who release the same copy of malware at m places simultaneously. We assume that the malware employs the localized random scanning strategy and the search spaces of the attackers are independent (e.g., attackers divide the whole IP address space equally into m parts and each attacker will be responsible for one part). For the initializations, we assume that x = 0 and y = 1 for all attackers. According to Equation (26), the propagation of the malware released by all attackers can be represented as $I_{0,1,j}$ because their initial values and β coefficients are the same. Note that $|\beta|$ = $\frac{kV}{c(\frac{2^{32}}{m}-b)}$ now since the search space for each attacker is now reduced to $\frac{2^{32}}{m}$. Recall that we require $|\beta| < \frac{1}{4}$. As discussed in Section V-C, if we assume that V = 2^{20} and b = 0, we have $\frac{kV}{c(\frac{2^{32}}{m}-b)}$ = $\frac{mk}{2^{12}c} < \frac{1}{4}$. Hence, $\frac{mk}{c} < 2^{10}$, which means that the product of the number of threads per malware and the number of attackers divided by the number of ports scanned should be smaller than 1024, if there are around one million vulnerable hosts. We assume that this is true and denote the propagation of the whole collaborative attack as $ITOTAL_{x_{total},y_{total},j}$.

According to Corollary 1 of the the Linear Property, we have:

$$\begin{aligned} ITOTAL_{x_{total},y_{total},j} &= \sum_{n=0}^{m-1} I_{0,1,j} \\ &= mI_{0,1,j} \\ &= I_{0,m,j} \end{aligned}$$

Hence, the number of the infected hosts of the m uncoordinated attacks that perform localized scanning is equivalent to that of the single attack released at one point with initial values x_{total} = 0, and y_{total} = m.

Case 2. There are still m collaborative attackers releasing the malware. We assume that the malware employs the sophisticated scanning strategy (but each malware copy has the same search space) and the malware at different hosts can communicate with each other to avoid duplicate infection attempts. Note that we do not assume the infected hosts can avoid duplicate scanning (in which multiple attackers can be modeled as one attacker with a huge number of threads and minimized thread maintenance costs). We assume that the initial values of the propagation of the malware released by Attacker A_n are x_n and y_n (n∈[0..m)), respectively. We still denote the propagation of the whole collaborative attack as $ITOTAL_{x_{total},y_{total},j}$.

According to the Linear Property (Theorem 2) in Section III-D4, we have:

$$\begin{aligned} ITOTAL_{x_{total},y_{total},j} &= \sum_{n=0}^{m-1} I_{x_n,y_n,j} \\ &= I_{\sum_{n=0}^{1} x_n, \sum_{n=0}^{1} y_n, j} + \sum_{n=2}^{m-1} I_{x_n,y_n,j} \\ &= ... \\ &= I_{\sum_{n=0}^{m-1} x_n, \sum_{n=0}^{m-1} y_n, j} \end{aligned} \tag{32}$$

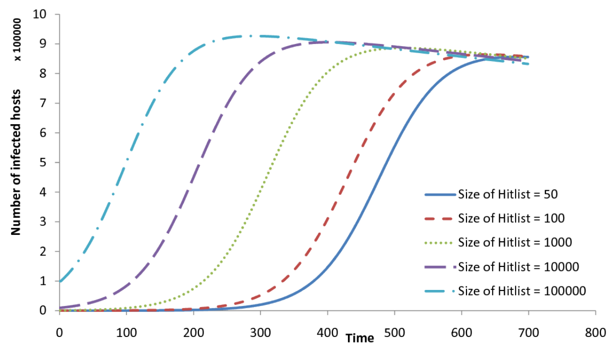

Fig. 2. The Propagation of the multi-threaded malware with different hitlist sizes.

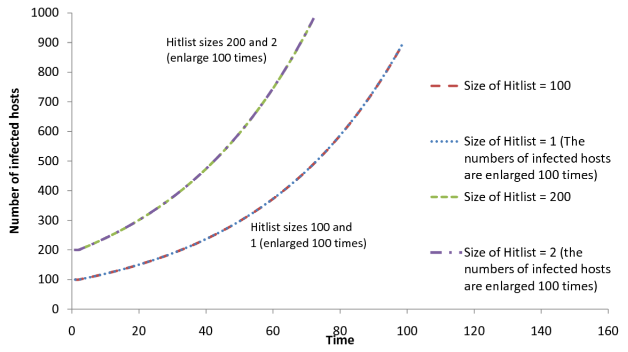

Fig. 3. The propagation of hitlist size 100 and 200/100 times of hitlist sizes 1 and 2.

Hence, the power of the m collaborative attacks is equivalent to the single attack released at one point with initial values x_{total} = $\sum_{n=0}^{m-1} x_n$, and y_{total} = $\sum_{n=0}^{m-1} y_n$. Equation (32) quantifies the power of collaborative attacks. It grows much faster than Equation (26).

VI. Experiments

In this section, we present the experiments on the GFMP model, including experimental setup, methods, and results. We have conducted the experiments on a Pentium 4 3.2GHZ workstation with 1GB physical memory and the Gentoo Linux Operating System. In all the experiments, we set V (the number of all vulnerable hosts) to 1,000,000, c (the number of ports the malware scans for one host) to 1, and b (the number of IP addresses that the malware does not scan) to 2^{25} (approximately the local and multicast addresses) in the GFMP model.

A. The effect of different hitlist sizes on the multi-threaded propagation

We conducted experiments to evaluate whether the hitlist scanning can accelerate the propagation of multi-threaded malware, and compare the effects of different hitlist sizes on the malware propagation. In this experiment, we set k (the number of threads in the vulnerability scanner of the malware) to 100, d (destruction rate) to 0.0001, p (patching rate) to 0.0002, and r (birth rate) to 0. Note that we set birth rate to 0 to show the effects of threading (otherwise the increased number of infected hosts may be caused by the newly joined vulnerable hosts).

Fig. 2 shows the malware propagation with hitlist sizes 50, 100, 1000, 10000, and 100000. We observe that the propagation speed of the multi-threaded malware increases as the size of the hitlist increase. Specifically, with the hitlists of sizes 100000, 10000, 1000, 100, and 50, the malware propagated to 500,000 hosts in 100, 210, 319, 441, and 489 time ticks (seconds), respectively. We conclude that the hitlist scanning can effectively accelerate the multi-threaded malware propagation, especially when the size of the hitlist is very large.

When the size of the hitlist is 100,000, the malware propagation reached its peak after 290 time ticks, after which the actual number of infected hosts decreased. The decrease is caused by the patching and destruction. In our experiment, the destruction and patching rates are not zero. Moreover, we set the birth rate to zero so that there will be no new vulnerable hosts to join the network. Therefor, after the malware propagation reached its peak, there will be no more vulnerable hosts to infect, and the patched hosts can no longer be infected. Hence, the number of infected hosts decreases afterwards. We note that different birth rates and patching rates can cause different propagation behaviors. We discuss our experimental results on the birth rates and patching rate in Section VI-D and Section VI-E, respectively.

B. Verification of the Shift Property and modeling of the Witty Worm

We performed experiments to verify the Shift Property presented in Section V-D2 and to model the propagation of the notorious Witty Worm. We set k to 100, d to 0, p to 0.0002, and r to 0.0002 so that the conditions for the property are satisfied.

From Equation (31), $GI_{0,h,j} = hI_j$, we conclude that the the number of infected hosts with hitlist size h divided by the number of infected hosts with hitlist size 1 is h. Hence, when the sizes of the hitlists are 2, 100, and 200, the quotients are 2, 100, and 200, respectively. We observe that the number of infected hosts with hitlist size 200 should be $\frac{200}{100}$=2 times of the number of infected hosts with hitlist size 100.

Fig. 3 shows the results on the propagation with hitlist sizes 1, 2, 100, and 200. Note that the numbers of infected hosts for hitlist sizes 1 and 2 are enlarged 100 times. The results confirm our analysis, and verify the Shift Property. For instance, we observe that the numbers of infected hosts with hitlist size 100 (or 200) are essentially coincident with the numbers of infected hosts (enlarged 100 times) with hitlist size 1 (or 2). The numbers of infected hosts with hitlist size 200 are approximately twice as many as the numbers of infected hosts with hitlist size 100.

According to Equation (30), $GI_{0,h,j} = I_{[j+\frac{\log(h\phi)}{\log\phi}-1]}$. Therefore, we can calculate the number of shifts required to calculate the number of infected hosts with hitlist size 100. Since k = 100, V = 2^{20}, b = 2^{25}, c = 1, we have: q = $\frac{100*2^{20}}{1*(2^{32}-2^{25})}$ = $\frac{100}{32*127}$ = 0.025. Hence, $\theta = \sqrt{1+4q} = \sqrt{1.098} = 1.048$,

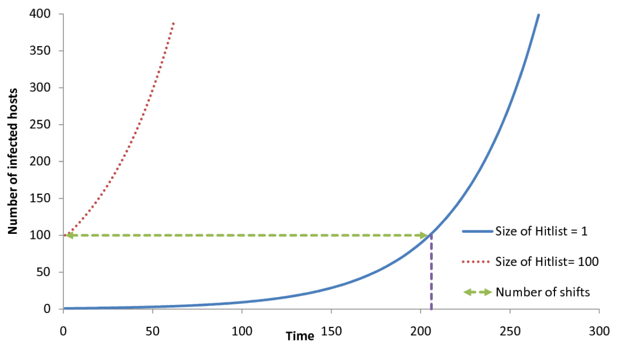

Fig. 4. The propagation of hitlist size 100 and 200/100 times of hitlist sizes 1 and 2.

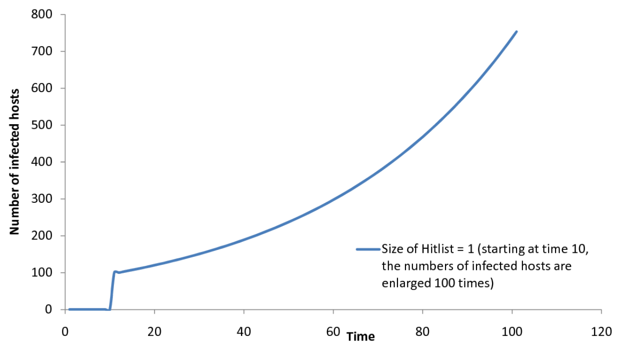

Fig. 6. The Propagation predicted by Equation 30 and 31 (100 times hitlist = 1 and shift by 10 time ticks.

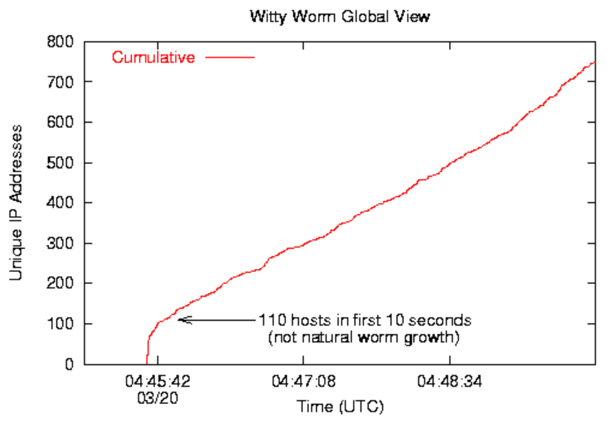

Fig. 5. The Real-World Propagation of the Witty Worm (from caida.org).

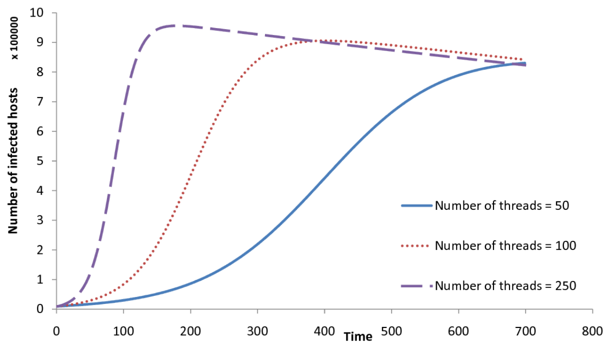

Fig. 7. The malware propagation with different number of threads.

and $\phi = \frac{1+\theta}{2} = 1.024$ Therefore, the number of shifts is : $\frac{\log((h)\phi)}{\log\phi} - 1 = \frac{log(100*\phi)}{log\phi}$ -1 = $\frac{2.010}{0.010}$ - 1 = 200.

Fig. 4 confirms our analysis. The solid line shows the propagation with hitlist size 1. The dotted line shows the propagation with hitlist size 100. The dashed line that connects the solid line and the dotted line illustrates the number of shifts required. We observe that the number of shifts required is approximately constant. The projection of the dashed line on the x-axis suggests that the number of shifts is 206. Therefore, the analytical result deviates from the experimental results by only $\frac{206-200}{206} = 2.9\%$.

Moreover, we compare our experimental results to the real-world propagation of the Witty Worm. The Witty Worm [2] is suspected to employ a hitlist scanning or was released on previously hacked hosts. Fig. 5 (Figure 1, Witty Worm Global View, in [2], from www.caida.org) produced by C. Shannon and D. Moore shows the initial spread (the first half minute) of the Witty Worm. The initial spread is unusual and could not be explained by the existing models including the AAWP model, because the worm propagated to 110 hosts in the first 10 seconds.

However, our GFMP model correctly shows that the effect of hitlist scanning can be approximated by the linear combination of two random scannings. When the linear Fibonacci Coefficient (h, the size of hitlist) is large, the unusual growth of the malware propagation can be explained by the linear combination of regular propagations. Fig. 6 shows our experimental results with hitlist size 1 (enlarged 100 times, and right-shifted by 10 time ticks). We observe that the graph approximately matches the propagation of the Witty Worm. Subtle differences exist (e.g., the concavity), and possible causes are different operating environment and different settings of parameters (e.g., k and b).

C. The effect of different threading-levels

We conducted experiments to identify the relationship between the number of scanning threads in the malware and its propagation. In this experiment, we set d to 0.0001, p to 0.0002, r to 0, and the hitlist size to 10000.

Fig. 7 shows the propagation with different threading-levels: 50, 100, and 250. We observe that the propagation speed increases as the number of threads in the malware increases. The effects of the multi-threads are significant: when the number of threads is 50, the malware took almost 700 time ticks to infect 800,000 hosts, while the same malware took approximately 300 time ticks with 100 threads and 100 time ticks with 250 threads to accomplish the same task. Note that when the number of threads is 250, the malware propagation reaches its peak in less than 200 time ticks. The number of infected hosts then decreased because of the destruction and the patching. Since we assume that a patched host cannot be infected again in this experiment, the number of infected hosts keeps decreasing afterwards. The patching rate we set in this experiment is fairly high (0.0002), which means that in every time tick two out of one thousand infected hosts are patched.

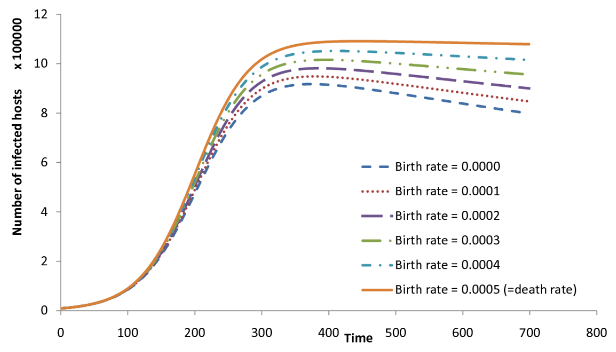

Fig. 8. The malware propagation with different birth rates.

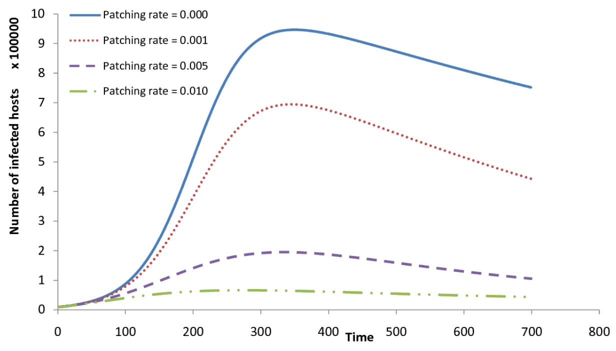

Fig. 9. The malware propagation with different patching rates.

D. The effect of different birth rates

We conducted experiments to study the effects of different birth rates on the malware propagation. In this experiment, we set d to 0.0005, p to 0.0000, k to 100, and the hitlist size to 10000. Note that we set the patching rate to 0 to focus on the birth rates.

Fig. 8 shows the propagation with birth rates 0, 0.0001, 0.0002, 0.0003, 0.0004, and 0.0005. Note that when the birth rate is 0.0005, it is equal to the destruction rate (0.0005). We observe that the birth rate does matter during the malware propagation. Specifically, we observe that the number of the infected hosts peaked at 1050000 when the birth rate is 0.0004, while the number of the infected hosts peaked at only 917000 when the birth rate is 0.

Moreover, when the birth rate is 0.0005, which is equal to the destruction rate, we observe that the number of infected hosts peaked at 1080000. The number of infected hosts neither increased nor decreased afterwards. Therefore, an equilibrium was reached: although 5 out of 10000 hosts are destructed in each time tick, 5 out of 10000 hosts are newly born and infected by the malware in each time tick.

E. The effect of different patching rates

We performed experiments to evaluate the effects of different patching rates on the malware propagation. In this experiment, we set d to 0.0001, r to 0.0003, k to 100, and the hitlist size to 10000.

Fig. 9 shows the malware propagation with patching rates ranging from 0 to 0.010. We observe that patching significantly reduces the number of infected hosts. Specifically,

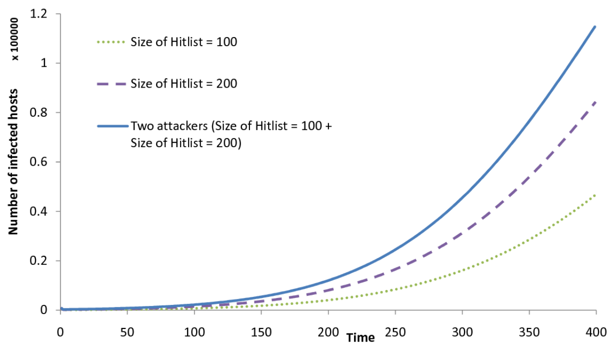

Fig. 10. The malware propagation with multiple attackers.

we observe that the malware propagation peaked at 900,000 hosts when there was no patching, and the number dropped to approximately 700,000 hosts when the patching rate was just 0.001, which means that only one out of one thousand hosts is patched. Furthermore, we see that the malware propagation peaked at only 193,000 hosts when the patching rate was 0.005, and the malware propagation was significantly reduced and peaked at only 66,000 hosts when the patching rate was 0.01 (one out of one hundred hosts). Therefore, we conclude that patching can significantly diminish the malware propagation and should be employed by most networks, if possible.

F. The effect of multiple attackers

We conducted experiments to study the effects of multiple attackers on the malware propagation. In this experiment, we set d to 0.005, p to 0.005, r to 0.03, and k to 100. We first set the hitlist size to 100 and 200, respectively, and performed the experiments. Then, we simulated the multiple-attack scenario discussed in Case 2 of Section V-D3: there are two collaborative attackers, one with hitlist size 100, and the other with hitlist size 200. The two attackers start at the same time, and communicate each other to avoid duplicate infection attempts.

According to Equation (32), the effects of the collaborative attack is the sum of the individual attacks. Fig. 10 presents the experimental results. The dotted line represents the propagation with hitlist size 100. The dashed line represents the propagation with hitlist size 200. The solid line represents the propagation with two attackers, one with hitlist size 100 and the other with hitlist size 200. We observe that the number of infected hosts for the collaborative attack is approximately the sum of the number of hosts infected for the individual attacks with hitlist sizes 100 and 200 initially, which proves Equation (32). However, we note that after around time tick 300, the sum of the number of infected hosts for the individual attacks become larger than the number of infected hosts for the collaborative attacks, which means:

$$ITOTAL_{x_{total},y_{total},j} < I_{\sum_{n=0}^{m-1} x_n, \sum_{n=0}^{m-1} y_n, j}$$

The explanation is that the number of infected hosts for the collaborative attacks decreased faster due to the possible contention between the collaborating attackers. In Case 2 of

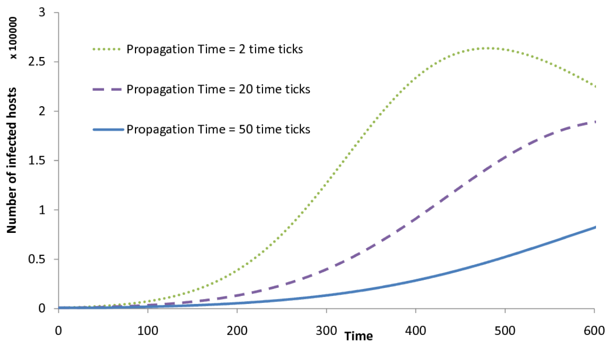

Fig. 11. The malware propagation with different propagation time.

Section V-D3, we assume that the collaborative attackers can avoid duplicate infection attempts, but cannot avoid duplicate scanning. Hence, some scanning activities in the collaborative attack may collide and such collision can cause the reduced efficiency of the collaborative attack.

G. *The effect of different propagation times*

We performed experiments to evaluate the effects of different propagation times on the malware propagation. In this experiment, we set d to 0.005, p to 0.005, r to 0.003, k to 100, and the hitlist size to 1000. Note that in our experiments, the destruction rate and the patching rate are high. The sum of the two rates are 0.005 + 0.005 = 0.01. We note that the birth rate is 0.003, which is smaller than the patching rate.

Fig. 11 shows the malware propagation with different propagation times: 2 time ticks, 20 time ticks, and 50 time ticks. We observe that as the propagation time increases, the propagation speed decreases. In 200 time ticks, the malware infected 38416, 13249, and 5512 hosts, with the 2-time-tick, the 20-time-tick, and the 50-time-tick propagation times, respectively. In 300 time ticks, the malware infected 125980, 39497, and 13403 hosts, with the 2-time-tick, the 20-time-tick, and the 50-time-tick propagation times, respectively. Furthermore, we observe that the malware propagation reached its peak at time tick 482 with 263,732 infected hosts with the 2-time-tick propagation time, while the malware propagation with the 20-time-tick and the 50-time-tick propagation times are still in the process of trying to infect more nodes. Therefore, we conclude that the defenders to malware should try to maximize its propagation time (e.g., by throttling [18]).

VII. Conclusion

In this paper, we present the Generic Fibonacci Malware Propagation (GFMP) Model to address the issues of multi-port scanning, multi-threading, infection time, multiple start points, and collaborative attacks in the malware propagation. We consider the multi-threading issue in the calculation of the probability of successful scans. We propose the propagation tree and propagation forest, and model the infection time and propagation time of the malware. In the GFMP model, we effectively utilize the generic Fibonacci Number Sequence (FNS), mathematically infer several important properties of the generic FNS, and apply them to quantitatively analyze the malware propagation. In particular, we analyze the effects of the propagation time, multiple starting points, and collaborative attacks. We perform experiments to verify our model. The experimental results confirm our mathematical analysis, and demonstrate that the GFMP model fit into the real-world propagation data, including the propagation of the notorious Witty Worm.

In Case 2 of Section V-D3, we assume that the malware at different hosts can communicate with each other to avoid duplicate infection attempts. We note that the communication may incur overhead, which could be caused by network delays, the limits of communication protocols, and the sizes of the data buffers at different hosts, etc. Currently the overhead is not represented in the GFMP model. Moreover, to improve the efficiency of scanning, the malware should not scan the already infected machines. A key observation is that data structures similar to the hash table or the DHT (Distributed Hash Table) [7] can be applied in this case to efficiently answer whether the IP address was already scanned or not. More advanced algorithms used in routers may be applied [33]. Advanced malware can also employ intelligent localized-scan algorithms. Enhancing the GFMP model with representation of communication and processing overhead, intelligent collaboration schemes, and smart localized-scan algorithms for the malware is the subject for future work.

Researchers discussed how to improve the performance of scanning by sampling [3]. With the prevalence of wireless networks, there will be more dynamic hosts that may join and leave the network frequently. We plan to extend the GFMP model to represent the sampling scheme and the dynamic host memberships.

In Section V, we assume that the propagation time is the same for all infections, and derived Equation (25) and (27). In the real-world, the propagation time for different infections may vary. For example, if the propagation time is three time ticks, we can apply the Tribonacci Number Sequence [34] to study the malware propagation. Extending the GFMP model to analyze the effects of the varying propagation time is the subject for future work.

In this paper, we focused on applying the Fibonacci Number Sequence (FNS) and its properties to analyze the performance of malware propagation schemes, including collaborative malware propagation. We employed the patching rate to model the defense activities as a black box. Questions then arise as how to defend sophisticated malware and delay the collaborative propagation. Modeling and analysis of the sophisticated collaborative defense is an interesting subject for future work.

Acknowledgement

The authors would like to thank Ziyu Zhang at Stanford University and Yi Mao at Purdue University for helpful discussions. This work was supported in part by grants from NSF-0242840 and NSF-0219110.

References

[1] Z. Chen, L. Gao, and K. Kwiat, Modeling the Spread of Active Worms, Proceedings of the IEEE INFOCOM, 2003

[2] http://www.caida.org/research/security/witty/, last accessed Apr 20, 2008

[3] M. Vojnovic, V. Gupta, T. Karagiannis, and C. Gkantsidis, Sampling Strategies for Epidemic-Style Information Dissemination, Proceedings of the IEEE INFOCOM, 2008

[4] S. Sarat and A. Terzis, Measuring the Storm Worm Network. Technical Report 01-10-2007, http://hinrg.cs.jhu.edu/uploads/Main/STORMTR.pdf

[5] C. Kanich, K. Levchenko, and B. Enright, G. M.Voelker and S. Savage, The Heisenbot Uncertainty Problem: Challenges in Separating Bots from Chaff, Proceedings of the USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET), San Franciso, CA, Apr. 2008

[6] Z. Chen and C. Ji, Optimal Worm-Scanning Method Using Vulnerable-Host Distributions International Journal of Security and Networks: Special Issue on Computer and Network Security, vol. 2, 2007

[7] R. Vogt, J. Aycock, and M. Jacobson, Army of Botnets, Proceedings of the Network and Distributed System Security Symposium (NDSS), San Diego, CA, Feb. 2007

[8] A. Svensson, A note on generation times in epidemic models, Mathematical Biosciences, Vol. 208, Iss. 1, Jul. 2007

[9] C. Zou, D. Towsley, and W. Gong, On the Performance of Internet Worm Scanning Strategies, Elsevier Journal of Performance Evaluation, Jul. 2006

[10] C. Zou, W. Gong, and D. Towsley, Code Red Worm Propagation Modeling and Analysis, 9th ACM Conference on Computer and Communication Security (CCS'02), Washington DC, Nov. 2002

[11] D. Moore, C. Shannon, and J. Brown, Code-Red: a case study on the spread and victims of an Internet Worm. In Proc. ACM/USENIX Internet Measurement Workshop, France, November, 2002

[12] A. Kamra, H. Feng, V. Misra and A. Keromytis, The Effect of DNS Delays on Worm Propagation in an IPv6 Internet, Proceedings of the IEEE INFOCOM, 2005.

[13] J. Yang, Fast Worm Propagation in IPv6 Networks, http://www.cs.virginia.edu/ jy8y/publications/cs85104.pdf

[14] http://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers

[15] S. Staniford, V. Paxson and N. Weaver, How to Own the Internet in Your Spare Time, In Proceedings of the 11th USENIX Security Symposium, Aug. 2002

[16] B. Wiley, Curious Yellow: The First Coordinated Worm Design, http://blanu.net/curious_yellow.html, Last accessed Apr 20, 2008

[17] J. Wu, S. Vangala, L. Gao, and K. Kwiat, An Effective Architecture and Algorithm for Detecting Worms with Various Scan Techniques, Proceedings of the Network and Distributed System Security Symposium (NDSS), 2004

[18] J. Twycross and M. Williamson: Implementing and Testing a Virus Throttle. In Proceedings of the 12th USENIX Security Symposium, Washington, 2003

[19] M. Vivo, E. Carrasco, G. Isern, and G. Vivo, A review of port scanning techniques, ACM Computer Communications Review, Vol. 29, Apr. 1999

[20] A.G. Voyiatzis and D.N. Serpanos, Pulse: A Class of Super-Worms against Network Infrastructure. ICDCS Workshops 2003

[21] J. Jung, V. Paxson, A.W. Berger, and J. Balakrishnan, Fast Portscan Detection Using Sequential Hypothesis Testing, In Proc. of the IEEE Symposium on Security and Privacy, May 2004

[22] S. Staniford, J.A. Hoagland, and J.M. McAlerney, Practical Automated Detection of Stealthy Portscans. Journal of Computer Security, 10, 2002

[23] S. Bellovin, B. Cheswick, and A. Keromytis, Worm propagation strategies in an IPv6 Internet, LOGIN, Vol 31. No.1.

[24] A. Wagner, T. Dubendorfer, B. Plattner, and R. Hiestand, Experiences with Worm Propagation Simulations, ACM Workshop on Rapid Malcode (WORM), 2003

[25] http://nmap.org/book/man-performance.html

[26] Kademlia Specification http://xlattice.sourceforge.net/components/protocol/kademlia/specs.html

[27] Z. Chen and C. Ji, A Self-Learning Worm Using Importance Scanning, ACM CCS Workshop on Rapid Malcode (WORM), 2005

[28] A. Kumar, V. Paxson, and N. Weaver, Exploiting Underlying Structure for Detailed Reconstruction of an Internet-scale Event. In the proceedings of ACM IMC, New Orleans, LA, Oct. 2005.

[29] C. Gates, Coordinated Port Scans: A Model, A Detector and An Evaluation Methodology. PhD Thesis. Dalhousie University. Feb. 2006

[30] S. Friedl, Analysis of the new Code Red II Variant, http:// www.unixwiz.net/ techtips/CodeRedII.html, Last accessed Apr 15, 2008

[31] T. Koshy, Fibonacci and Lucas Numbers with Applications, Wiley-Interscience, Aug. 2001

[32] Editorial, A Cyberblockade in Estonia, New York Times, Jun. 2, 2007

[33] M.A. Ruiz-Sanchez, E.W. Biersack, and W. Dabbous, Survey and taxonomy of ip address lookup algorithms, IEEE Network Magazine, Vol. 15, Mar. 2001

[34] I. Dumitriu, On generalized Tribonacci sequences and additive partitions, Discrete Mathematics, 219, 65-8, 2000

PLACE PHOTO HERE

Yu Zhang received the B.E. degree in Computer Science from Special Class for Gifted Young, University of Science and Technology of China and the M.S. degree in Computer Sciences from Purdue University. He is currently working towards the Ph.D. degree at Purdue University. His research interests include collaboration, distributed systems, and security. He has worked at Cisco Systems Inc. and Google Inc. Research.

PLACE PHOTO HERE

Bharat Bhargava received the B.E. degree from the Indian Institute of Science and the M.S. and Ph.D. degrees in E.E. from Purdue University. He is a professor of computer science at Purdue University. His research involves adaptability, wireless network, and secure routing. He has been awarded the charter Golden Core Member distinction by the IEEE Computer Society for his distinguished service and, in 1999, he received an IEEE Technical Achievement Award for the major impact of his decade-long contributions to the foundations of adaptability in communication and distributed systems. He serves the IEEE Computer Society on Technical Achievement Award and Fellow committees. He is the founder of the IEEE Symposium on Reliable and Distributed Systems (SRDS), IEEE conference on Digital Library, and the ACM Conference on Information and Knowledge Management (CIKM). He is a fellow of the IEEE and the IEEE Computer Society.